

Experiment 1

1. The lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Develop a lexical Analyzer to identify identifiers, constants, operators using C program.

Program:

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int main()
{
    int i,ic=0,m,cc=0,oc=0,j;
    char
b[30],operators[30],identifiers[30],constants[30];
    printf("enter the string : ");
    scanf("%[^\\n]s",&b);
    for(i=0;i<strlen(b);i++)
    {
        if(isspace(b[i]))
        {
            continue;
        }
        else if(isalpha(b[i]))
        {
            identifiers[ic] =b[i];
            ic++;
        }
        else if(isdigit(b[i]))
        {
            m=(b[i]-'0');
```

```

i=i+1;
while(isdigit(b[i]))
    {
        m=m*10 + (b[i]-'0');
        i++;
    }
i=i-1;
constants[cc]=m;
cc++;
}
else
    {
        if(b[i]=='*')
            {
                operators[oc]='*';
                oc++;
            }
        else if(b[i]=='-')
            {
                operators[oc]='-';
                oc++;
            }
        else if(b[i]=='+')
            {
                operators[oc]='+';
                oc++;
            }
        else if(b[i]=='=')
            {
                operators[oc]='=';
                oc++;
            }
    }

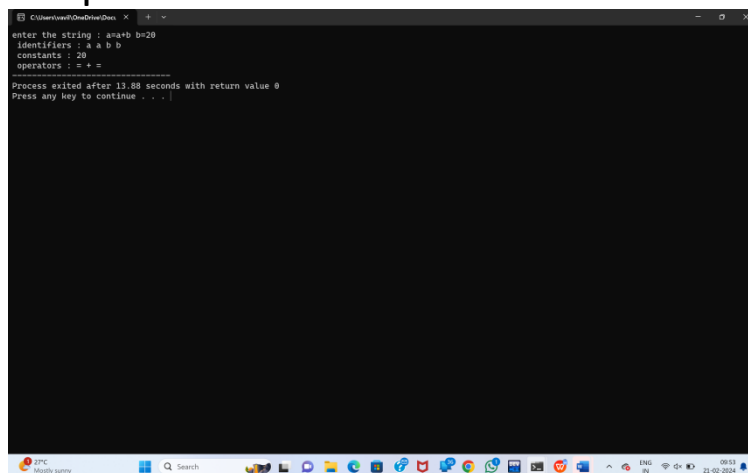
```

```

    }
    }
}
printf(" identifiers : ");
for(j=0;j<ic;j++)
{
    printf("%c ",identifiers[j]);
}
printf("\n constants : ");
for(j=0;j<cc;j++)
{
    printf("%d ",constants[j]);
}
printf("\n operators : ");
for(j=0;j<oc;j++)
{
    printf("%c ",operators[j]);
}
}

```

Output:



```

C:\Users\user\OneDrive\Desktop >
enter the string : anab b=20
identifiers : a a b b
constants : 20
operators : + + =
=====
Process exited after 13.88 seconds with return value 0
Press any key to continue . . .

```